**《Java技术》实验报告**

实验二： 2020年 04月 10 日

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | 计算机与信息学院 | 专业班级 | 物联网工程19-2班 | 姓名 | 袁焕发 | 成绩 |  |
| 课程  名称 | Java技术 | 实验项目  名 称 | 货物进销管理系统 | | | 指导 教师 | 薛峰 |
| 教师评语 | 教师签名：  年 月 日 | | | | | | |
|  | 一、实验目的  1．掌握Java中文件的读写操作。  2．学会使用Java提供的实用类（Vector, ArrayList）来完成特定的功能。  3．掌握字符串类（String, StringBuffer）的使用。  4．掌握用面向对象的方法分析和解决复杂问题。  二、实验原理  编写一个Inventory.java完成以下功能（没有学过Java文件处理之前，各位同学可以使用硬编码将数据放进两个Vector变量里。等学过Java文件处理之后，再补充数据文件读取部分）：  1．程序首先打开并读取Inventory.txt中记录的所有库存记录，然后读取Transactions.txt，处理这个文件中包含的事务，记录发货记录到Shipping.txt，并记录错误信息到Errors.txt中。最后更新库存到另外一个文件NewInventory.txt中。  2．文件Inventory.txt和NewInventory.txt的每行包含一个存货记录，没条记录包含下面一些字段息，这些字段之间用一个tab分开（见后面的文件格式）：     |  |  | | --- | --- | | 字段 | 格式和含义 | | Item number | 字符串型，货物编号 | | Quantity | 整型，货物数量 | | Supplier | 字符串型，供应商编号 | | Description | 字符串型，货物描述 |   3．字段Items按照从小到大的顺序写入文件的。注意Item号不必连续，如Item号为752的后面可能是800。  4．文件Transactions.txt包含几个不同的处理记录（每行一条记录）。每条记录前面以一个大写字母开头，表示这条记录是什么类型的事务。在不同的大写字母后面是不同的信息格式。所有的字段也是以tab键分开的（见Transactions.txt文件格式）。  5．以'O'开头的事务表示这是一个发货订单，即某一种货物应该发给特定的客户。Item number和Quantity的格式如上面表格定义。Custom编号和上面的Supplier编号一致。处理一条定单记录（以'O'开头的事务）意味着从减少库存记录中相应货物的数量（减少的数量=发货单中的数量），记录发货信息到Shipping.txt中。注意：Inventory.txt中的quantity不应该小于0，如果对于某一种货物，库存的数量小于发货单的数量的话，系统应该停止处理发货单，并记录出错信息到Errors.txt。如果对于某一种货物有多个发货单，而且库存总量小于这些发货单的总和的话，系统应该按照发货单中的数量从小到大的有限原则满足客户。也就是说，对于某一种货物如果一个数量Quantity少的发货单没有处理之前，数量Quantity多的发货单永远不会被处理。（这种处理原则不受发货单记录在Transactions.txt的先后顺序影响）  6．以'R'开头的事务表示这是一个到货单记录，在'R'后面是Item number和它的数量Quanlity。处理一条到货单意味着增加库存中相应货物的数量（增加的数量=到货单中的数量）。注意：如果在Transactions.txt文件中，到货单出现在发货单之后，到货单中的货物数量可以用来填补发货单中的数量（可以理解成在Transactions.txt中，优先处理到货单）。  7．以'A'开头的事务表示向库存中增加一种新的货物（即这种货物以前库存中没有），在'A'后面是Item number，供应商supplier以及货物的描述description。处理一个新增货物记录意味着向库存中增加一个数量Quantity为0的新的Item。你可以假设在一个Transactions.txt中，新增货物记录总是出现在第一个到货单之前。  8．以'D'开头的事务表示从库存中删除一种货物，在'D'后面是Item号。删除操作总是在所有的事物处理之后才被处理，以保证对于可能出现的同一种货物的发货单的操作能在删除之前被正确处理。如果要删除的某种货物的库存量Quantity不为0的话，系统应该向Errors.txt记录出错信息。  9．文件Shipping.txt中的每一行代表给某一客户的发货信息。Shipping.txt中的每一行分别是客户编号、Item号、货物数量，它们之间用tab键分隔。如果发货单中有两条客户编号和Item编号一样的记录，在Shipping.txt中应该将这两条发货信息合并（即将它们的数量相加）。  10．Errors.txt文件包含未发送的发货记录和库存量大于0的删除记录。Errors.txt每一行包含Custom编号、Item编号以及发货单上的数量Quantity。对于删除操作，Custom编号为0，数量Quntity为库存中的Quantity.  11．实验测试数据:  Inventory.txt    Transactions.txt | | | | | | |
|  | 三、使用硬件、软件环境  PC 计算机一台，配置为CPU为2.4G，内存为16G，硬盘为500G，安装Windows10操作系统。  另外，使用Notepad++，Eclipse ,JDK1.8.0等软件  四、实验过程、步骤及原始记录(算法、原程序、测试结果，分析等)  首先在Inventory.java中分别构建不同的类对应不同的操作，实例化类的对象，并构建动态数组进行不同信息的存储。再分别对不同的具体操作要求进行函数的编写，最后在主函数进行函数的调用  源代码  package testwork2;  import java.io.\*;  import java.util.\*;  public class Inventory {    String itemnum; //货物编号  int quantity;//数量  String supplier; //供应商  String desc; //描述  //构造函数  public Inventory(String itemnum, int quantity, String supplier, String desciption) {    this.itemnum = itemnum;  this.quantity = quantity;  this.supplier = supplier;  this.desc = desciption;  }  Inventory(Add tmp) { //增加货物就要存入数组  itemnum = tmp.itemnum;  quantity = 0;  supplier = tmp.supplier;  desc = tmp.desc;  }    static Vector<Inventory> inv = new Vector<Inventory>();  static Vector<Order> order = new Vector<Order>();  static Vector<Order> shipping = new Vector<Order>();  static Vector<Recv> itemrecv = new Vector<Recv>();  static Vector<Add> itemadd = new Vector<Add>();  static Vector<Delete> itemdelete = new Vector<Delete>();  static Vector<Err> err = new Vector<Err>();  public static void main(String[] args) throws Exception {  readinve("D:\\code\\testtxt\\Inventory.txt");  readTran("D:\\code\\testtxt\\Transactions.txt");  doAdd(); //添加新货物  doReceive(); //收货入库  arrOrder(); //订单从小到大排序  doOrder(); //出货  doDelete();//删除数量为0货物  arrshipping(); //去除重复  write("D:\\code\\testtxt\\NewInventory.txt D:\\code\\testtxt\\Shipping.txt D:\\code\\testtxt\\Errors.txt");  }  public static void readinve(String filename) throws Exception {  BufferedReader br = new BufferedReader(new FileReader(filename));  String line = "";  while ((line = br.readLine()) != null) {  String tmp[] = line.split("\t"); //遇到tab分割  int quantity = Integer.parseInt(tmp[1]);  //解析一个字符串，并返回一个整数  inv.add(new Inventory(tmp[0], quantity, tmp[2], tmp[3]));  }  br.close(); //关闭输入流  }  public static void readTran(String filename) throws Exception {  BufferedReader br = new BufferedReader(new FileReader(filename));  String line = "";  while ((line = br.readLine()) != null) {  String[] tmp = line.split("\t");  //遇到tab就分割字符串  if (tmp[0].equals("O")) {  //如果是发货记录  int quantity = Integer.parseInt(tmp[2]);  order.add(new Order("O", tmp[1], quantity, tmp[3]));    } else if (tmp[0].equals("R")) {  //如果是收货记录  int quantity = Integer.parseInt(tmp[2]);  itemrecv.add(new Recv("R", tmp[1], quantity));    } else if (tmp[0].equals("A")) {  //如果是新增货物记录  itemadd.add(new Add("R", tmp[1], tmp[2], tmp[3]));    } else if (tmp[0].equals("D")) {  //如果是删除记录  itemdelete.add(new Delete("D", tmp[1]));  }  }  br.close();  }  //添加货物  public static void doAdd() {  for (int i = 0; i < itemadd.size(); i++) {  //小于要添加货物的数目时不断添加到货物数组  Inventory tmp = new Inventory(itemadd.get(i));  inv.add(tmp);  }  }  //进行收货入库  public static void doReceive() {  //收到货物的数量，加到原有货物上  for (int i = 0; i < itemrecv.size(); i++) {  for (int j = 0; j < inv.size(); j++) {  String s1, s2;  s1 = new String(itemrecv.get(i).itemnum);  s2 = new String(inv.get(j).itemnum);  if (s1.equals(s2)) {  (inv.get(j)).quantity+=((itemrecv.get(i)).quantity);  }  }  }  }  //出货，  public static void doOrder() {  for (int i = 0; i < order.size(); i++)  for (int j = 0; j < inv.size(); j++) {  String s1, s2;  s1 = new String(order.get(i).itemnum);  s2 = new String(inv.get(j).itemnum);  if (s1.equals(s2)) {  if ((inv.get(j)).quantity>= (order.get(i)).quantity) {  //货物充足可以出货，减去出货量  (inv.get(j)).quantity-=((order.get(i)).quantity);  shipping.add(order.get(i));  } else {  //货物不足无法出货，输出错误信息  Err tmp = new Err(order.get(i));  err.add(tmp);  }  }  }  }    //删除  public static void doDelete() {  for (int i = 0; i < itemdelete.size(); i++) {  for (int j = 0; j < inv.size(); j++) {  String s1, s2;  s1 = new String(itemdelete.get(i).itemnum);  s2 = new String(inv.get(j).itemnum);  if (s1.equals(s2)) {  if ((inv.get(j)).quantity == 0) {  //如果货物剩余为0，则删除货物  inv.remove(j);  break;  } else {  //否则输出  err.add(new Err((inv.get(j)).itemnum,  (inv.get(j)).quantity,  (inv.get(j)).supplier));  break;  }  }  }  }  }  //订单排序将小的订单放在前面  public static void arrOrder() {  for (int i = 0; i < order.size(); i++)  for (int j = i + 1; j < order.size(); j++) {  int qua1=order.get(i).quantity;  int qua2=order.get(j).quantity;  String sup1=order.get(i).supplier;  String sup2=order.get(j).supplier;  String item1=order.get(i).itemnum;  String item2=order.get(j).itemnum;  //如果出货数量多则排在后面  if ((order.get(i)).quantity > (order.get(j)).quantity) {  order.get(i).supplier=sup2;  order.get(i).quantity=qua2;  order.get(i).itemnum=item2;  order.get(j).supplier=sup1;  order.get(j).quantity=qua1;  order.get(j).itemnum=item1;    }  }  }  //合并重复的记录  public static void arrshipping() {  for (int i = 0; i < shipping.size(); i++)  for (int j = i + 1; j < shipping.size(); j++) {  String item1, item2, sup1, sup2;  item1 = new String(shipping.get(i).itemnum);  item2 = new String(shipping.get(j).itemnum);  sup1 = new String(shipping.get(i).supplier);  sup2 = new String(shipping.get(j).supplier);  //如果货物编号和供应商相同就删除  if (item1.equals(item2) && sup1.equals(sup2)) {  shipping.remove(j);  j--;  }  }  }  public static void write(String filenames) throws Exception {  String[] filename = filenames.split(" ");  BufferedWriter invbw = new BufferedWriter(new FileWriter(filename[0]));  BufferedWriter orderbw = new BufferedWriter(new FileWriter(filename[1]));  BufferedWriter errbw = new BufferedWriter(new FileWriter(filename[2]));  String a="Item编号 货物数量 供应商 货物描述";  invbw.write(a);  invbw.newLine();  for (int i = 0; i < inv.size(); i++) {  Inventory tmp = inv.get(i);  String s = tmp.itemnum + "\t" + tmp.quantity + "\t" + tmp.supplier + "\t" + tmp.desc;  invbw.write(s);  invbw.newLine(); //另起一行  }  String aString="客户编号 Item号 货物数量";  orderbw.write(aString);  orderbw.newLine();  for (int i = 0; i < shipping.size(); i++) {  Order tmp = shipping.get(i);  String s = tmp.supplier + "\t" + tmp.itemnum + "\t" + tmp.quantity;  orderbw.write(s);  orderbw.newLine();  }  errbw.write(aString);  errbw.newLine();  for (int i = 0; i < err.size(); i++) {  Err tmp = err.get(i);  String s = tmp.supplier + "\t" + tmp.itemnum + "\t" + tmp.quantity;  errbw.write(s);  errbw.newLine();  }  invbw.close();  orderbw.close();  errbw.close();  }      }  class Order {  String type;  String itemnum;  int quantity;  String supplier;  public Order() {  }  //以'O'开头的事务表示这是一个发货订单  public Order(String Type, String Itemnum, int Quantity, String Supplier) {    Type="O";  this.type=Type;  this.itemnum=Itemnum;  this.quantity = Quantity;  this.supplier = Supplier;  }  public void copy(Order o) {  this.itemnum=o.itemnum;  this.type="O";  this.quantity = o.quantity;  this.supplier = o.supplier;  }  }  // 收货记录  //以'R'开头的事务表示这是一个到货单记录，在'R'后面是Item number和它的数量quanlity  class Recv {  String type;  String itemnum;  int quantity;  public Recv(String Type, String Itemnum, int Quantity) {  Type="R";  this.type=Type;  this.itemnum=Itemnum;  this.quantity = Quantity;  }  }  // 添货记录  class Add {  String type;  String itemnum;  String supplier;  String desc;  int quantity;  public Add(String Type, String Itemnum, String Supplier, String Desc) {  Type="A";  this.type=Type;  this.itemnum=Itemnum;  this.supplier = Supplier;  this.desc = Desc;  }    }  // 删货记录  class Delete {  String type;  String itemnum;  public Delete(String Type, String Itemnum) {  Type="D";  this.type=Type;  this.itemnum=Itemnum;  }  }    // 错误记录  class Err {  String itemnum;  int quantity;  String supplier;  Err(String Itemnum, int Quantity,String Supplier) {  this.itemnum = Itemnum;  this.quantity = Quantity;  this.supplier = Supplier;  }  Err(Order o) {  this.supplier = o.supplier;  this.itemnum = o.itemnum;  this.quantity = o.quantity; }  }    五、实验结论、分析、思考题与心得体会   1. 学会了动态数组的使用，利用动态数组来储存信息，对于数组内成员信息的处理有了更加熟练的操作。 2. 学会了文件的按路径读写，知道了利用缓冲流BufferedReader和BufferedWritter可以一次性读取和写入多行字符。 3. 学到了利用split方法对字符串进行分割，利用get方法获得数据，利用parseInt将字符型解析为整型。 4. 学会了利用面向对象，构造不同的类来对不同的问题进行处理，对于类的构造和初始化和类中方法的调用编写有了更深的认识。 | | | | | | |